13주차 결과보고서

전공: 컴퓨터공학과 학년: 4학년 학번: 20192135 이름: 윤영인

**1. 실습**

예비보고서에서 재귀로 Pseudo code를 작성하였지만, 구현에 문제가 있어 노드와 스택 구조체를 만들어 DFS를 구현하였다.

**1.1 구조체**

1) Node 구조체

텍스트, 폰트, 스크린샷, 라인이(가) 표시된 사진

자동 생성된 설명

스택을 구성하는 노드 구조체를 생성하였다. r과 c 변수는 미로의 좌표를 의미하며 연결리스트로 노드를 잇기 위해 next 포인터를 정의하였다.

2) MyStack 구조체

텍스트, 폰트, 라인, 그래프이(가) 표시된 사진

자동 생성된 설명

스택 구조체이다. 포인터 top 변수는 스택의 top을 의미하며, len은 스택 안의 노드의 개수를 의미한다.

3) allPath 구조체

텍스트, 폰트, 스크린샷, 라인이(가) 표시된 사진

자동 생성된 설명

스택으로 탐색한 모든 경로와 탈출 지점까지 잇는 경로를 찾아 각각 연결리스트로 표현하기 위해 선언한 구조체이다. 탐색한 좌표 지점의 노드가 들어가며 head 포인터 변수는 연결리스트의 첫 시작 부분을 의미한다.

**1.2 makeNode()**

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

스택을 구성하는 노드를 만드는 함수이다. 각 지점의 좌표 값을 인자로 받으며, Node 타입의 메모리를 동적으로 할당한다. 할당한 노드 공간에 인자로 받아온 좌표 값을 넣어주고 next 포인터에 NULL을 넣고 리턴한다.

**1.3 makeStack()**

텍스트, 스크린샷, 폰트, 라인이(가) 표시된 사진

자동 생성된 설명

스택을 생성하는 함수이다. MyStack 타입의 메모리 공간을 동적할당한 후, top엔 NULL, len엔 0을 넣어 초기화하여 리턴한다.

**1.4 isEmpty()**

텍스트, 폰트, 스크린샷, 라인이(가) 표시된 사진

자동 생성된 설명

스택이 비어 있는지 확인하는 함수이다. len이 0이면 비어있으므로 true를, 그렇지 않으면 false를 리턴한다.

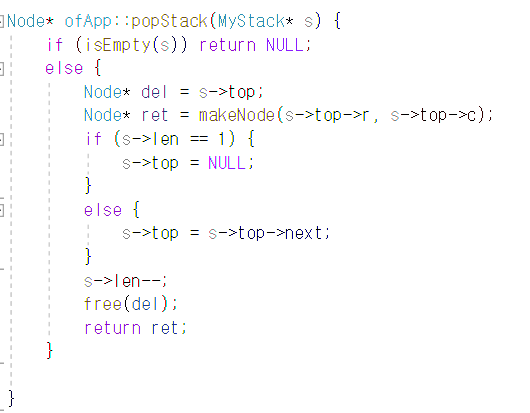
**1.5 pushStack()**

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

스택에 노드를 push하는 함수로, 스택이 비어있으면 top에 새로운 노드를 넣어준다. 비어있지 않으면 스택의 특성인 후입선출을 만족시키기 위에 현재 top 변수 위에 새로운 노드를 추가하고 스택의 top을 새로 들어온 노드를 가리키도록 한다. 스택을 모두 넣어줬으면 len을 1 증가한다.

**1.6 popStack()**



스택에서 팝하는 함수로, 스택은 후입 선출이므로 top 노드르 팝한다. 만약 스택이 비어 있으면 NULL을 반환하고 비어 있지 않으면, 리턴하기 위해 현재 스택의 탑 노드의 좌표를 넣어 새로운 노드를 생성한다. 만약 현재 스택의 길이가 1이면 탑 노드에 NULL을 넣어주고, 그렇지 않으면 현재 탑 노드를 그 다음 노드로 바꾼다. 스택의 길이를 1 줄이고 현재 탑 노드를 넣어준 ret 노드 변수를 free하고, 위에서 탑 노드의 좌표를 넣어 생성한 노드를 반환한다.

**1.7 makePath()**

텍스트, 스크린샷, 폰트, 라인이(가) 표시된 사진

자동 생성된 설명

스택으로 탐색한 모든 경로와 탈출 지점까지 잇는 경로를 찾아 넣기 위한 연결리스트를 생성하는 함수이다. 연결리스트를 allPath타입의 메모리 공간으로 동적할당하고 head 부분에 NULL을 넣어준 후 리턴한다.

**1.8 DFS()**

텍스트, 스크린샷, 번호, 폰트이(가) 표시된 사진

자동 생성된 설명

DFS로 길을 찾기 위해 탐색하는 방향은 남동북서로 정하였다. 시작 지점인 (1,1) 위치를 노드에 담아 스택에 넣어준다. 탐색한 모든 경로를 넣어주는 연결리스트인 ap와 최단 경로를 넣어주는 sp 변수를 추가로 선언하여 초기화한다. 스택이 비지 않을 동안 탐색을 진행하는데, 만약 현재 탑 노드의 좌표가 미로의 탈출 지점이면 반복문을 break로 끝낸다. 그렇지 않으면 현재 위치에서 남, 동, 북, 서 차례로 1칸씩 이동하면서 탐색을 진행한다. 만약 탐색 지점이 미로 안에 있고 아직 방문하지 않은 이동할 수 있는 방이면, 스택에 넣어준다. 방문 체크를 하고, 탐색한 모든 경로를 넣어주는 ap에 현재 위치를 넣어준다. 이 지점은 현재 위치에서 이동이 가능하므로 move 변수를 true로 넣어준 후 for 반복문을 빠져나온다.

텍스트, 스크린샷, 디스플레이, 폰트이(가) 표시된 사진

자동 생성된 설명

반복문을 빠져나오거나 끝났을 때, move 변수가 false이면 현재 위치에서 이동이 불가능하다는 것이므로 스택에서 pop한다. 최종적으로 반복문이 모두 끝났을 때, 스택엔 탈출 지점까지의 이동 경로가 담겨 있으므로 sp에 스택의 노드들을 담아준다.

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

탐색한 모든 경로가 들어있는 ap에 노드를 추가하는 함수이다. 만약 ap가 비어 있으면 head에 새로운 노드를 넣어주고, 그렇지 않으면 가장 끝 노드를 찾아 새로운 노드를 넣어준다.

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

미로의 시작부터 탈출 지점을 잇는 경로를 담은 연결리스트에 경로를 넣어주는 함수이다. addPath와 같은 방식으로 구현하였다.

DFS 함수는 스택이 빌 동안 while문을 실행하며, popStack과 pushStack에서 O(1)의 시간이 필요하므로 최종적으로 이 함수의 시간 복잡도는 O(V)로, V는 노드의 개수이다. 스택과 경로를 저장하는 ap와 sp 변수를 위해 메모리 공간이 필요하며, 둘다 노드의 수에 비례하므로 공간 복잡도는 O(V)로, V는 노드의 개수이다.

**1.9 dfsdraw()**

텍스트, 스크린샷, 번호, 폰트이(가) 표시된 사진

자동 생성된 설명

탈출 지점까지 방문한 위치와 시작~탈출 경로를 그려주는 함수이다. 우선 방문한 모든 위치를 그리기 위해 색은 검정색으로 지정하였고, ap의 연결리스트의 모든 노드를 순회하여 ofDrawRecTangle로 노드에 담긴 좌표에서 weight 크기만큼 색이 채워진 직사각형을 그렸다. 여기서 weight은 20으로 헤더파일에 선언하였다. 시작~탈출 경로 역시 색이 채워진 직사각형으로 그리되 색은 빨간색으로 지정하였다. 이것도 역시 sp의 연결리스트의 모든 노드를 순회하면서 직사각형을 그리도록 구현하였다. 이 함수의 시간 복잡도는 모두 연결리스트의 노드의 수에 비례하므로 O(n)이다. 이 함수에선 별도의 메모리 공간을 할당하지 않으므로 공간 복잡도는 O(1)이다.

**1.10 결과물**

직사각형, 스크린샷, 도표, 폰트이(가) 표시된 사진

자동 생성된 설명

위의 함수들로 실행한 결과물이다.

**2. 과제**

**2.1 구조체**

1) MyQueue

텍스트, 폰트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

큐를 사용하여 BFS를 구현하기 위해 큐 구조체를 선언하였다. front와 rear 포인터 변수는 큐 구조체에서 front와 rear를 의미하고, len은 큐에 담긴 노드의 개수를 의미한다.

2) Pos

텍스트, 폰트, 스크린샷, 라인이(가) 표시된 사진

자동 생성된 설명

큐에서 미로를 탐색할 때 이전 위치를 저장하기 위한 구조체로, 위치의 좌표를 담기 위해 r, c 변수를 저장하여 선언하였다.

**2.2 makeQueue**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

큐를 생성하는 함수로, MyQueue 타입의 메모리를 동적으로 할당하여 front와 rear에 nullptr과 len에 0을 넣어 초기화한 후 리턴한다.

**2.3 isQueueEmpty()**

텍스트, 폰트, 스크린샷, 라인이(가) 표시된 사진

자동 생성된 설명

큐가 비었는지 확인하는 함수로 isEmpty 함수와 마찬가지로 len이 0이면 true, 아니면 false를 반환하도록 하였다.

**2.4 pushQueue()**

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

큐에 새로운 노드를 push하는 함수로, 선입선출인 큐의 특성에 맞도록 구현하였다. 큐가 비어있으면 front와 rear에 새로운 노드를 넣었다. 비어있지 않으면 선입선출을 하도록 rear 위로 새로운 노드를 넣고 새로운 노드로 rear를 업데이트 하였다. 큐에 노드를 추가하였으면 len을 1 증가시켰다.

**2.5 popQueue()**

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

큐에서 노드를 팝하는 함수로, front를 팝하도록 구현하였다. 큐가 비어 있으면 NULL을 반환하고, 그렇지 않으면 front를 리턴하기 위해 front의 좌표를 담은 새로운 노드를 생성한다. 만약 큐의 길이가 1이면 front와 rear에 nullptr을 넣고, 그렇지 않으면 front를 그 다음 노드로 변경한다. front를 free하고 큐의 길이를 1 줄인 후 front의 좌표를 담아 생성한 노드를 리턴한다.

**2.6 BFS()**

BFS를 실행하기 전에 DFS에서 사용한 visited배열을 0으로 초기화하고, isdfs 변수를 false로 변경해야한다. 아래의 사진은 visited 배열을 초기화하고, 변수들을 false로 변경하는 코드이다.

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명 텍스트, 스크린샷, 폰트, 친필이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 디스플레이, 폰트이(가) 표시된 사진

자동 생성된 설명

큐를 탐색하면서 이전 좌표를 저장하기 위해 Pos 타입의 2차원 배열을 선언하고 -1로 초기화하였다. DFS 함수와 마찬가지로 ap와 sp에 방문한 모든 경로 및 시작~끝 경로를 넣어주었다. 시작 지점을 큐에 넣어주고 방문 체크를 한다.

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

큐가 비지 않을 동안 while문을 돌면서 탐색을 시작한다. 큐에서 front를 팝하여 팝한 지점이 탈출 지점일 경우 while문을 빠져나온다. 그렇지 않으면 현재 위치에서 남, 동, 북, 서 차례로 한 칸씩 이동하여 움직일 수 있는 방인지 확인한다. 만약 아직 방문하지 않은 이동가능한 방일 경우 방문 체크를 해주고 큐에 담는다. 그리고 prev 배열의 이동한 위치에 현재 좌표 값을 넣어 이전 경로를 저장하도록 한다.

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

bfs는 탐색하면서 탈출 지점에 도착했을 경우가 가장 최단 경로이므로, 이전 경로가 담긴 배열을 확인하면서 최단 경로를 저장하는 sp에 경로를 담도록 하였다. pR과 pC에 탈출 지점에 담긴 이전 좌표를 담아서 시작 지점이 될 때까지 경로를 역추적하여 업데이트하도록 하였다. 이 과정은 최단 경로를 따라 이전 노드를 거슬러가면서 역추적하는 함수이다. 최단 경로를 sp에 담았으면 큐와 prev 배열을 할당 해제한다.

이 함수의 시간 복잡도는 큐의 크기에 비례하므로 O(V+E)이며, 이때 V는 노드의 수, E는 간선의 수를 의미한다. 공간 복잡도는 prev 배열 및 노드의 개수에 비례하므로 O(N^2) 이다.

**2.7 bfsdraw()**

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

탈출 지점까지 방문한 위치와 시작~탈출 경로를 그려주는 함수이다. 우선 방문한 모든 위치를 그리기 위해 색은 회색으로 지정하였고, ap의 연결리스트의 모든 노드를 순회하여 ofDrawRecTangle로 노드에 담긴 좌표에서 weight 크기만큼 색이 채워진 직사각형을 그렸다. 여기서 weight은 20으로 헤더파일에 선언하였다. 시작~탈출 경로 역시 색이 채워진 직사각형으로 그리되 색은 초록색으로 지정하였다. 이것도 역시 sp의 연결리스트의 모든 노드를 순회하면서 직사각형을 그리도록 구현하였다. 이 함수의 시간 복잡도는 모두 연결리스트의 노드의 수에 비례하므로 O(n)이다. 이 함수에선 별도의 메모리 공간을 할당하지 않으므로 공간 복잡도는 O(1)이다.

**2.8 결과물**

스크린샷, 텍스트, 직사각형, 도표이(가) 표시된 사진

자동 생성된 설명

위의 함수들로 실행한 결과물이다.

**3.**

미로 3주차에선 bfs와 dfs로 시작~탈출까지의 경로를 찾았다. DFS는 깊이를 하나씩 증가하면서 노드를 탐색하므로 탈출 경로가 깊은 높이에 있을 경우 유리하고 메모리 사용량이 적은 편이다. BFS는 최단 경로를 탐색할 때 유용하며, 주변 노드부터 탐색하므로 탈출 위치의 깊이가 깊지 않을 경우에 유리하다. 미로 찾기를 진행할 때 왼쪽 제일 위에서 오른쪽 가장 아래로 경로를 찾고 이동해야 한다. 따라서 탐색 방향을 남, 북, 동, 서로 정하였고 BFS와 비교했을 때 DFS는 남쪽부터 탐색하면서 이동 가능하면 해당 위치로 이동하여 다시 탐색하므로, 더 적합하다고 생각했다.